BILKENT UNIVERSITY

ENGINEERING FACULTY

DEPARTMENT OF COMPUTER ENGINEERING

CS342
Operating Systems
Project 3 Report

Furkan Kazım Akkurt 21702900
Ümit Yiğit Başaran 21704103

Spring 2020 - 2021

Contents

1 Introduction

There are several kinds of algorithms for efficient memory management systems. Those algorithms can be classified into static and dynamic memory management algorithms. Buddy algorithm is one of those algorithms that is classified as a dynamic memory management algorithm. In this report, firstly, the algorithm will be talked about, and then the results we obtained using the algorithm will be discussed in terms of fragmentation.

2 Buddy Algorithm

Buddy algorithm is an algorithm that uses dynamic allocation and deallocation. The goal of the algorithm is to allocate the most efficient size for a request by dividing memory into smaller chunks. The algorithm tries to use the efficiency of splitting memory into two halves. When a memory chunk is deallocated by a process, the deallocated segment will be merged into a larger segment with its buddy block, if the buddy block is available, i.e., is not used by any other process.

2.1 Implementation

For this project we are asked to design a shared memory which gives the memory chunks to the processes depending on request size and buddy algorithm. In our implementation, our shared memory contains, an integer value count ($sizeof(int)$) which represents the number of processes that are currently using the shared memory segment, an integer value order ($sizeof(int)$) which represents the maximum order (this value is used to represent the number of available node index), order + 1 available node ($(order+1)*sizeof(structavailablenode)$) which contains a long value head and a long value tail (these values are not taken as pointers because virtual memory locations of available memory blocks differs from process to process) and the requested/initialized shared memory segment (processes use this segment of the shared memory).

For each memory block, we have tag and order integer values ($sizeof(int)$). Tag value represents the availability of the block and the order value represents the order value of the block. These values are held in the block either when the block available or not. Available blocks has next and prev long values that represent the links to the other blocks that have the same order number and are available. Therefore, our whole structure is similar to the

linked list structure however links of the nodes are held in the exact blocks and the pointer values are not addresses rather than that they represents the distance values from the starting address of the shared segment so that it doesn't differ process to process.

3 Tests

In order to observe the results of the algorithm, different were conducted. Internal fragmentation is observed using the best case, worst case, and random case scenarios.

3.1 Worst Case Scenario

For our implementation, worst case occurs when the request size of the process is too close to the block size with order $k-1$ however it exceeds the size of it so that the system gives the block with order k. For instance, if the minimum block size is 128 byte and the programs requests 121 byte space, then, the system gives 256 byte memory space because in our implementation 128 byte has 8 byte offset and the best fit memory size is 120.

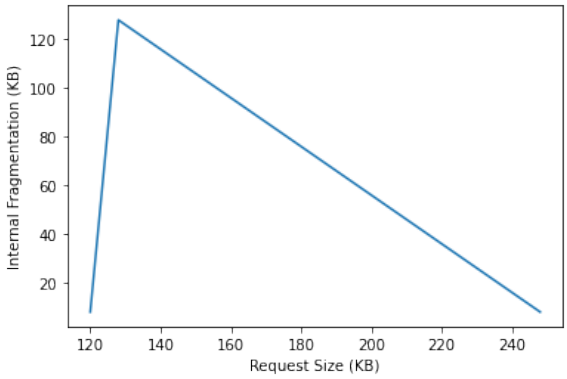

Figure 1: Worst Case Scenario Graph

As it can be seen from the figure 1, for our implementation the worst case occurs when the request size is too close to the previous block size but has greater request value. After the worst case the internal fragmentation is reduced because between the worst case to the block size the system gives the same size of memory space to the process.

3.2 Best Case Scenario

For our implementation, best case occurs when the request size is $(2^k - 8) * min\_block\_size$ where k is the order of the block. For an minimum block size of 64 byte, 56, 120, ... bytes of requests would be considered as best case scenario.

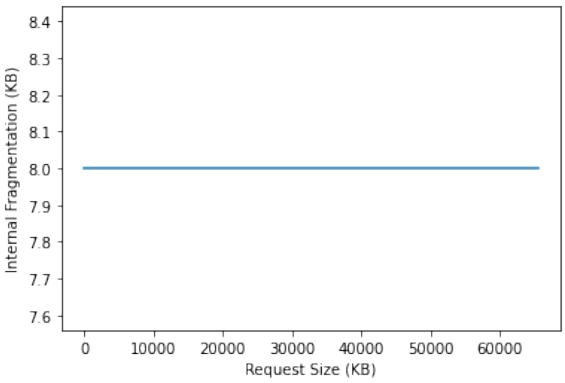

Figure 2: Best Case Scenario Graph

As it can be seen from the figure 2, best case scenario occurs when the request size is $(2^k - 8) * min\_block\_size$. In our implementation the minimum internal fragmentation value for one block is 8 byte because of the offset values (tag and order).

3.3 Random Case Scenario

Rather than requesting at worst or best case, the algorithm was also run using random sizes of requests.

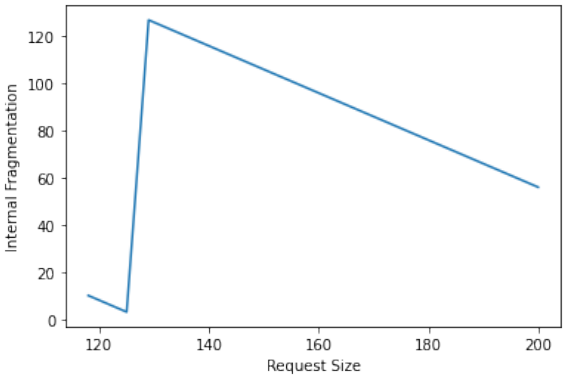

Figure 3: Random Case Scenario Graph

In figure 3, we give random request sizes in ascending order to the system and investigate the internal fragmentation values. By looking at the figure 3 we could say that the graph supports the inferences that we made in the worst and the best case scenarios.

4 Conclusion

Buddy memory management algorithm is a dynamic memory management system that tries to efficiently use the memory segments by dividing and merging the memory segments